The Imperative for High-Performance Audio Computing

John ffitch and **Richard Dobson** and **Russell Bradford**
Department of Computer Science
University of Bath
BATH BA2 7AY
United Kingdom
{jpff,rwd,rjb}@cs.bath.ac.uk

Abstract

It is common knowledge that desktop computing power is now increasing mainly by the change to multi-core chips. This is a challenge for the software community in general, but is a particular problem for audio processing. Our needs are increasingly towards real-time and low latency. We propose a number of possible paths that need investigation, including multi-core and special accelerators, which may offer useful new musical tools. We define this as High-Performance Audio Computing, or HiPAC, in analogy to current HPC activity, and indicate some on-going work.

Keywords

HiPAC, Parallel, Csound

1 Introduction

We have been particularly struck by the paper by James Moorer[1] where he states

> *This is compute power far beyond what even the most starry-eyed fortune-teller could have imagined! It will change the very nature of what audio is, and what audio engineers do, since it changes what is possible at a fundamental level.*

In pursuit of this aim we propose the topic of HiPAC — High-Performance Audio Computing — as the study of new advanced processor architectures to enhance audio synthesis, processing and music composition. The name is a deliberate echo of the field of general High-Performance Computing (HPC), but there are significant differences. Rather than considering the use of supercomputers and megaclusters we are concentrating on what will within a relatively short timescale be consumer-grade hardware. The emphasis has to be on affordable low latency real-time processing.

The long-established Moore's law, the doubling of the number of transistors in a chip every 18 months, has persisted as an indirect measure of processing power, so long as the emphasis has remained on increasing the processing power of a single CPU core. However, obtaining further speed from a single core (demanding not least, increased clock speeds) incurs a significant increase in power requirements, expressed not only in terms of wattage but also in terms of heat generation, which in turn demands further power for its dissipation using air or other cooling. This high power consumption is not only directly costly, it also implies an increasingly unacceptable "carbon cost", and for the musician, noise. The universally recognised solution is to increase the number of processing cores, so that more computations can be made concurrently. This move towards parallel computation has profound implications for both users and programmers. In this respect, music and audio processing raises a number of interesting challenges and opportunities. One the one hand, a digital audio stream is by definition serial, naturally suggesting a serial (if fast) mode of computation; on the other, practical audio processes may involve a large number of identical and often independent tasks which can in principle run concurrently.

For many musicians, the availability of higher computing power has typically been measured in terms of the number of demanding processes (such as reverberation, or synthesiser voices) that can be performed in real time. While this remains an important measure, we are interested more in the possibility of exploring in real time processes that have hitherto been disregarded as too computationally demanding but which the next generation of hardware could bring into to the realm of real-time execution.

2 The Hardware — from Supercomputers to the Desktop

For a period the interest was in "supercomputing", as typified by vector and array processors, like the original Cray 1, built in 1976 for the Los Alamos National Laboratory[2] and specified to reach the then astonishing speed of 160Mflops and an almost equally astonishing main memory of 8Mbytes. The machine solved a central issue for any cluster, of inter-node communication, by ensuring that no connection was longer than 4 feet. The resulting horseshoe shape became for the general public almost the defining iconic image of a “supercomputer”.

At a similar time in the UK there were many alternative designs, such as dataflow [3] and the ICL DAP[4].

There was a brief flowering of research into parallel computing, and much discussion on meshes and hyper-cubes. A key aspect was the use of a Single-Instruction-Multiple-Data (SIMD) computational mode, which is unsuited to computations distributed over physically disparate nodes. This leads to a number of synchronous processing elements, with a common clock, which must be physically very close – same board or chip. The concept of SIMD still dominates much computing, even in the SSE instructions in recent chips.

Early MIMD (Multiple-Instruction-Multiple-Data) processing was beset by communication costs and organisational issues. However it is this style that led the revival of interest, with the Beowulf cluster approach, using a network of commodity PCs.

In contrast, modern SIMD fine-grained parallel architectures are associated today firstly with the vector extensions to standard CPUs (*e.g.* Altivec on the PPC, SSE on x86) whereby several arithmetic units are employed in parallel, and secondly with the graphics accelerator cards now essential to all consumer workstations (especially where high performance in games is required). The difference clearly is that these SIMD systems are typically monolithic, implemented in one chip. An example of this approach is the IBM Cell Broadband Processor[5] employed in the Sony Playstation. This chip employs a conventional PowerPC master CPU coupled with 8 parallel floating-point PEs.

In the attempt to maintain a generational performance increase, chip manufacturers are already supplying devices and motherboards supporting multiple CPU cores (currently with a maximum of four cores per chip, in the case of the processors from Intel and AMD), while investigating designs which significantly increase the number of cores. Intel, for example, have publicised a development chip featuring 80 cores, and claim performance up to 1 Teraflop, while indicating that a commercial release may still be 10 years away. When associated with cluster computing we obtain the current batch of super computers – indeed it is now a mark of pride to have a cluster with more cores than ones neighbours.

At the same time a highly significant market has arisen for SIMD-style accelerator systems designed to work in conjunction with a host computer, and targeted at the HPC community. One significant example is the Tesla accelerator series from nVidia [6]. The company have for many years provided a SDK for several of their GPU-based video cards, featuring their CUDA development environment. The Tesla series represents their first general-purpose accelerator product not specifically developed as a graphics accelerator, while based on the same technology. The Graphics Processing Unit (GPU) has already attracted the interest of a number of audio researchers (*e.g.* [7]), especially for the computation of 3D acoustic modelling tasks. A second example, is the “Advance” floating-point accelerator card from the British startup Clearspeed[8]. Each card features two of their CSX600 chips, each offering 96 PEs supporting double-precision floating point computation, with very low power consumption of some 30W per card. Each card provides a sustained computation speed of 50 Gflops. Both the nVidia and Clearspeed products provide automatic acceleration support for Matlab, enabling them to be rapidly integrated into an existing HPC cluster.

Clearspeed’s most recent device, the CSX700, merges the two CSX600 cores onto one chip, and the new PCIe-based cards employing them are considerably cheaper, making them directly competitive with the nVidia Tesla product, and thus significantly more accessible to the individual user.

In a parallel development, FPGA devices have increased substantially in both size and speed, offering a powerful alternative path to custom DSP chip design for both academic research[9] and industry. For example, the new “Crystal Core” system from Fairlight is based on a dynamically reconfigurable FPGA device[10]. It is also of interest to note the use of special “physics” co-processors in some game computers.

3 The software

Over time there have been many attempts at software aimed at parallel execution schemes, often tied to particular hardware.

Of particular note is the now defunct Inmos Transputer, almost unique in being closely associated with a dedicated concurrent programming language Occam[11][1].

This in turn was based on the formal language Communicating Sequential Processes (CSP) de-

[1]An insider’s history of the Transputer can be found in [12; 13]

vised by Hoare[14], and which is still highly influential in the field of concurrent and parallel computation. Amongst computer musicians, the Transputer is of special note for its use in the first real-time parallel implementation of Csound[15], involving some 170 Transputers, though it is recounted that the problem of heat dissipation was never resolved. The Transputer however lives on in emulation, for example in the forms of an FPGA-based device[16] and of a software emulation of the instruction set supporting Occam[17].

There is a tension between using a language that directly supports a parallel and concurrent paradigm, and retaining the ubiquity of the C and C++ languages, neither of which embodies any explicit support for parallelism. In the latter case the trend is to a dependence on general threading libraries (such as POSIX pthreads for C, or the Boost C++ threading library), or on language enhancements such as OpenMP, that may or may not make explicit use of platform-specific facilities (with the many well-known issues associated with multi-threaded programming using these languages).

The prevailing trend is the emergence of a variety of custom extensions to C, usually serving particular hardware. For example, in a fashion similar to the MasPar, the Clearspeed hardware is supported by an extended C compiler supporting a custom keyword **poly** defining a variable stored (with unique values) on each of the 96 PEs and referenced as a single entity. Further extensions and library functions deal with the sometimes complex tasks of transferring data between PEs, and between **poly** and **mono** (conventional) memory. With respect to FPGA programming, the company Celoxica provides the language Handel-C (following CSP principles) enabling a range of FPGA development systems to be programmed using a similarly extended C language supporting both coarse and fine grained concurrency by means of **wait** and **par** keywords [18].

This very condensed and selective snapshot of the fields of hardware and software support for concurrent processing leads us to a consideration of the particular challenges presented by audio. In this regard, we must recall the inherently serial (one-dimensional) nature of raw audio data, which would seem to impose significant constraints on the full exploitation of parallel processing. Many of the fundamental processes in which we are interested, such as recursive filters, are inherently data-dependent and (taking the example of a plain first-order IIR filter) therefore by definition un-parallelisable. The relationship between sequential and parallel computation is summarised in Amdahl's law [19], which may be stated as the speedup being

$$1/(S + P/N)$$

where S is the fraction of serial computation, $P = 1 - S$ is the amount of parallelisable computation and N is number of processors.

The limiting value is therefore $1/S$ for an infinite number of processors; that is, computation will be dominated by the sequential subset. This law (which has been applied in such areas as business and project management as well as in computing) suggests some serious limits on how much speedup can be obtained from parallel processing. However, this has more recently been demonstrated to be an overly conservative estimate [20], with respect to a hypercube-based system. For audio processes an equivalent to Amdahl's law is as yet undefined, and would seem to be highly dependent on the characteristics of the architecture, as well as on the nature of the process itself. Clearly, a process comprised mostly of recursive processes will gain relatively little speedup (dependent entirely on the mapping of the number of recursive streams to the number of processors). On the other hand, many audio processes are inherently highly parallelisable with few or no data dependencies, most obviously frame-based analysis techniques such as the DFT. Such algorithms can be expected to derive the maximum benefit from parallel computation using SIMD models. As already indicated in the examples of graphic modelling, audio processes based on physical models (*e.g.* finite element networks) similarly offer a very good fit to a parallel computational model. One factor that must be borne in mind is that the clock speeds of such devices (rated in MHz rather than GHz), as also of most DSP chips, are often significantly lower than those currently employed in desktop PCs. The overall computation speed of such processors depends almost entirely on parallel computation, such that serial computation needs to be kept to an absolute minimum.

As the current technology as exemplified in the systems described above is seeking to reach and exceed 1 Teraflop speeds, we argue that it is now time to revisit audio processes hitherto disregarded on account of their computational demands. Even if they are time-consuming today, they will not be in only a few years, so that we *must* start investigating them now. By the same argument, we expect hardware that is currently relatively expensive to fall to commodity prices and therefore to become accessible to everyone. In particular, we advocate in the

HiPAC program the study of no-compromise algorithms – rather than make simplifications to an algorithm purely for reasons of slowness, HiPAC considers such algorithms in as pure or "ideal" a form as possible, especially where that ideal form may lead to musically useful and novel behaviour.

We present below two contrasting case studies which we have started.

We can summarise the primary defining characteristics of a pure HiPAC DSP process:

- use of highly parallel fine-grained architectures (*e.g.* following the SIMD model), though we do not exclude more "conventional" multi-core computation
- real-time performance or better
- implies low latency
- ideal and "no-compromise" forms of algorithms
- new processes, and hence new effects and sounds, not simply "more of the same" - whether more reverbs or more voices.

Finally we note that there are alternative forms of parallel and distributed computation that can be applied to audio, such as grids, web services[21] and clusters. We do not consider these further in this paper, but suspect they will eventually find a place in the galaxy.

4 A HiPAC case study – the Sliding Phase Vocoder

In the paper [22] we identified the The Sliding Phase Vocoder (SPV) as a canonical example of a HiPAC process. A fuller description of the SPV and its precursor SDFT can be found in that paper and elsewhere [23; 24; 25]

We focus here on the HiPAC aspects that relate to the SPV. The conventional phase vocoder transforms audio into the frequency domain by means of a series of overlapping Fourier transforms (using the FFT algorithm). All practical implementations (and especially where real-time performance is required) overlap analysis frames by some small fraction of the window size. For a given sample rate R, the analysis rate A is given by the overlap length D in samples, as $A = R/D$. This can be described as the "hopping phase vocoder". It is well known that increasing the overlap leads to improved sonic performance, but at a directly increasing computational cost. In the limit, as $D \rightarrow 1$, the "ideal" phase vocoder overlaps frames by one sample, so that $A = R$. While recognised in the literature as offering sonic advantages, this sliding form has hitherto been avoided in practice as being computationally prohibitive.

Our implementation of the SPV is based on the use of the Sliding DFT, in which the DFT frame is updated every sample by a simple complex rotation of the bin values

$$F_{t+1}(n) = (F_t(n) - f_t + f_{t+N})\, e^{2\pi i \frac{n}{N}}$$

It should be noted that this process is inherently parallel between the bins. It has also been found to reduce the latency by as much as 75% compared to conventional pvoc. The computational demands are greatly increased, it being in effect an N^2 process. However, most transformations applied to analysis frames are also parallelisable (often involving standard vector arithmetic operations).

We have also shown that pitch shifting is a much simpler process compared to the hopping pvoc, since the frequency range of each bin covers the whole audio range (resynthesis is by oscillator bank). This enables, for example, audio-rate Frequency Modulation to be applied cleanly to an arbitrary input, a process we have termed Transformational FM[22]. The single-sample update (permitting high modulation rates) is essential to the implementation of audio-rate FM, in both time and frequency domain forms; thus TFM is an example of a frequency-domain process that cannot be implemented using the hopping pvoc.

It should be noted that these algorithms are available in Csound5, but the dismal performance means that it is unlikely to be used in any but off-line processing.

In terms of the HiPAC criteria listed above, we can see that they are all met by the SPV:

- highly parallelisable
- streamable in real time given fast enough hardware
- lower latency compared to conventional pvoc
- an "ideal" or no-compromise version of pvoc
- enables a new class of transformation, TFM, not realisable using standard pvoc.

We are investigating the use of the Clearspeed accelerator chips in implementing a viable version of this algorithm. It has been shown that this hardware can support significantly in excess of a hundred oscillators, so seems like a good match.

5 A HiPAC case study – Parallelising Csound

A slightly different case is exemplified by Csound. There is a multitude of music pieces that depend on the syntax and semantics of the Csound system, and it is a cardinal feature of the system that existing pieces must not be broken by developments[26]. If this synthesis language is to work in a world where everyone have a multicore computer we need to find a way to harness multiple processes. It is simple to use one processor for the GUI and one for the processing, but this is hardly sufficient. We need to split the audio processing itself between processors or threads.

There have been at least three previous attempts at a parallel Csound. We have already cited the Transputer Array project. A similar project was Midas[27], where the processing was developed as a flowgraph of unit generators, and then mapping them to a network of SGI workstations. This process has an inherent latency, and the communication costs tend to dominate. It would be good to experiment with this scheme, but it seems unavailable now.

Another less documented system is that of Vercoe, where Extended Csound was implemented on a small number of SHARC processes, with a fixed allocation of instruments to processors, and a rendezvous point at the end of each kontrol cycle. This scheme is also found in the later work described at Sounds Electric[28]. The model restricts the language, and introduces a change in semantics relating to the order of instruments, but it is a working system.

A full solution needs to take account of granularity and semantics. The parcels of work need to be of sufficient size to overcome the communication costs, and the semantics defining the order of instruments and global variables needs analysis. We previously were involved in a parallel simulation project in LISP[29] where we followed the process [30] of analysing the program for points of synchronisation, and also evaluated the possible cost of individual functions[31]. We are currently investigating a compiler-technology approach to the parallelisation of csound, using the new parser as the basis of the flowgraph, and hence of the identification of synchronisation, and using valgrind and some scripts to count the cost of individual ugens. This work is reported in more detail in [32].

6 Conclusions

We define HiPAC as a new class of computationally demanding audio processes implemented by means of the next generation of parallel processing platforms and tools. We have described the primary trends with respect both to desktop computers and to hardware accelerators. The latter are already offering close to Teraflop-class computing power. We expect the cost of these devices to drop significantly within the next decade, so that parallel computing will move from the domain of HPC to the home desktop. We have presented the Sliding Phase Vocoder as a canonical example of a HiPAC process. Many other audio processes are known that are well suited to parallel implementation. We also outlined a contrasting parallel audio problem, utilising multicore processors with exiting applications. We would also cite in particular processes based on physical models (whether of instruments or of acoustic spaces), which can be realised not only with optimised waveguides, but also in a more "ideal" form using finite element models. We suggest that by implementing these computationally intensive processes now, we prepare the ground for the immediate exploitation of the next generations of parallel computing hardware, which may be with us much sooner than we might have supposed only a year or two ago.

This paper is a revision of the paper[33] presented at ICMC2009, revised in the light of comments and a panel discussion.

References

[1] James A. Moorer. Audio in the New Millennium. *J. Audio Eng. Soc.*, 48(5):490–498, May 2000.

[2] Cray History. `http://www.cray.com/about_cray/history.html`, 2008.

[3] J. R. Gurd and D. F. Snelling. Manchester data-flow: a progress report. In *ICS '92: Proceedings of the 6th international conference on Supercomputing*, pages 216–225, New York, NY, USA, 1992. ACM.

[4] P. M. Flanders, D. J. Hunt, S. F. Reddaway, and D. Parkinson. Efficient high speed computing with the Distributed Array Processor. In David J. Kuck, Duncan H. Lawrie, and Ahmed H. Sameh, editors, *High Speed Computer and Algorithm Organization*, pages 113–128. Academic Press, 1977.

[5] IBM. Cell Broadband Engine resource center. `http://www.ibm.com/developerworks/power/cell`, 2007.

[6] NVIDIA Tesla GPU Computing Solutions for HPC. `http://www.nvidia.com/object/tesla_computing_solutions.html`, 2008.

[7] Niklas Röber, Ulrich Kaminski, and Maic Masuch. Ray Acoustics Using Computer Graphics Technology. In *Proc. of DAFx07*, pages 117–124, Bordeaux, France, September 2007.

[8] ClearSpeedTechnology. ClearSpeed Whitepaper: CSX Processor Architecture. `http://www.clearspeed.com`, Feb 2007.

[9] Markus Pfaff, David Malzner, Johannes Seifert, Johannes Traxler, Horst Weber, and Gerhard Wiendl. Implementing Digital Audio Effects Using a Hardware/Software Co-Design Approach. In *Proc. of DAFx07*, pages 125–132, Bordeaux, France, September 2007.

[10] `http://www.fairlightau.com`, 2008.

[11] INMOS, editor. *The occam2 Programming Manual*. Prentice-Hall, 1988.

[12] Iann Barron. Inmos and the transputer (part 1). *Computer Resurrection*, 32, New Year 2004. ISSN0958 - 7403.

[13] Iann Barron. Inmos and the transputer (part 2). *Computer Resurrection*, 33, Spring 2004. ISSN0958 - 7403.

[14] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[15] Nicholas. J. Bailey, Alan Purvis, Peter D. Manning, and I. W. Bowler. An Implementation of Csound on The Transputer. *Applications of Transputers*, 1, 1989.

[16] John Jakson. R16: A New Transputer Design for FPGAs. In Jan F. Broenink, Herman W. Roebbers, Johan P.E. Sunter, Peter H. Welch, and David C. Wood, editors, *Communicating Process Architectures 2005 – WoTUG-28*, volume 63 of *Concurrent Systems Engineering Series*. IUOS Press, 2005.

[17] Christian L. Jacobsen and Matthew C. Jadud. The Transterpreter: A Transputer Interpreter. In Dr. Ian R. East, Prof David Duce, Dr Mark Green, Jeremy M. R. Martin, and Prof. Peter H. Welch, editors, *Communicating Process Architectures 2004*, volume 62 of *Concurrent Systems Engineering Series*, pages 99–106. IOS Press, Amsterdam, September 2004.

[18] `www.celoxica.com`, accessed 2008.

[19] Gene M. Amdahl. *Validity of the single processor approach to achieving large scale computing capabilities*, pages 79–81. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[20] John L. Gustafson. Reevaluating Amdahl's law. *Commun. ACM*, 31(5):532–533, 1988.

[21] John ffitch, James Mitchell, and Julian Padget. Composition with sound web services and workflows. In Suvisoft Oy Ltd, editor, *Proceedings of the 2007 International Computer Music Conference*, volume I, pages 419–422. ICMA and Re:New, August 2007. ISBN 0-9713192-5-1.

[22] Russell Bradford, Richard Dobson, and John ffitch. The sliding phase vocoder. In Suvisoft Oy Ltd, editor, *Proceedings of the 2007 International Computer Music Conference*, volume II, pages 449–452. ICMA and Re:New, August 2007. ISBN 0-9713192-5-1.

[23] Russell Bradford, Richard Dobson, and John ffitch. Sliding is Smoother than Jumping. In SuviSoft Oy Ltd, Tampere, Finland, editor, *ICMC 2005 free sound*, pages 287–290. Escola Superior de Música de Catalunya, 2005. `http://www.cs.bath.ac.uk/~jpff/PAPERS/BradfordDobsonffitch05.pdf`.

[24] John ffitch, Richard Dobson, and Russell Bradford. Sliding DFT for Fun and Musical Profit. In Frank Barknecht and Martin Rumori, editors, *6th International Linux Audio Conference*, pages 118–124, Kunsthochscule für Medien Köln, March 2008. LAC2008, Tribun EU, Gorkeho 41, Bruno 602 00. ISBN 978-80-7399-362-7.

[25] Russell Bradford, Richard Dobson, and John ffitch. Sliding with a Constant Q. In *Proc. of the Int. Conf. on Digital Audio Effects (DAFx-08)*, pages 363–369, Espoo, Finland, Sep 1-4 2008. DAFx08. ISBN 978-951-22-9517-3.

[26] John ffitch. The Design of Csound5. In *LAC2005*, pages 37–41, Karlsruhe, Germany, April 2005. Zentrum für Kunst und Medientechnologie. `http://www.cs.bath.ac.uk/~jpff/PAPERS/ffitch05.pdf`.

[27] Tim Anderson, Andy Hunt, Ross Kirk, P. McGilly, Richard Orton, and Sean Watkinson. From Score to Unit Generator – A hierarchical view of MIDAS. In *Proc. International Computer Music Conference*, pages 235–238, 1992.

[28] Barry Vercoe. Keynote speech. Presented at Sounds Electric, Maynooth, April 1905.

[29] C. Burdorf, J. P. Fitch, J. B. Marti, and J. A. Padget. A Multiprocessor Execution Profiler. In Bruce D. Shriver, editor, *Software*

Track Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, pages 524–531, 1989.

[30] J. P. Fitch and J. B. Marti. The Bath Concurrent LISP machine. In *Proceedings of EUROCAL 1983*, volume 162 of *Lecture Notes in Computer Science*, pages 78–90, 1984.

[31] J. P. Fitch and J. B. Marti. The Static Estimation of Runtime. Technical Report 89–18, University of Bath Computing Group, 1989.

[32] John ffitch. A Method for Parallel Csound. 2009. Submitted for publication.

[33] Richard Dobson, John ffitch, and Russell Bradford. High Performance Audio Computing – A Position Paper. In *Proceedings of the 2008 ICMC*, pages 213–216, SARC, Belfast, 2008. ICMA and SARC. ISBN 0-9713192-6-x.